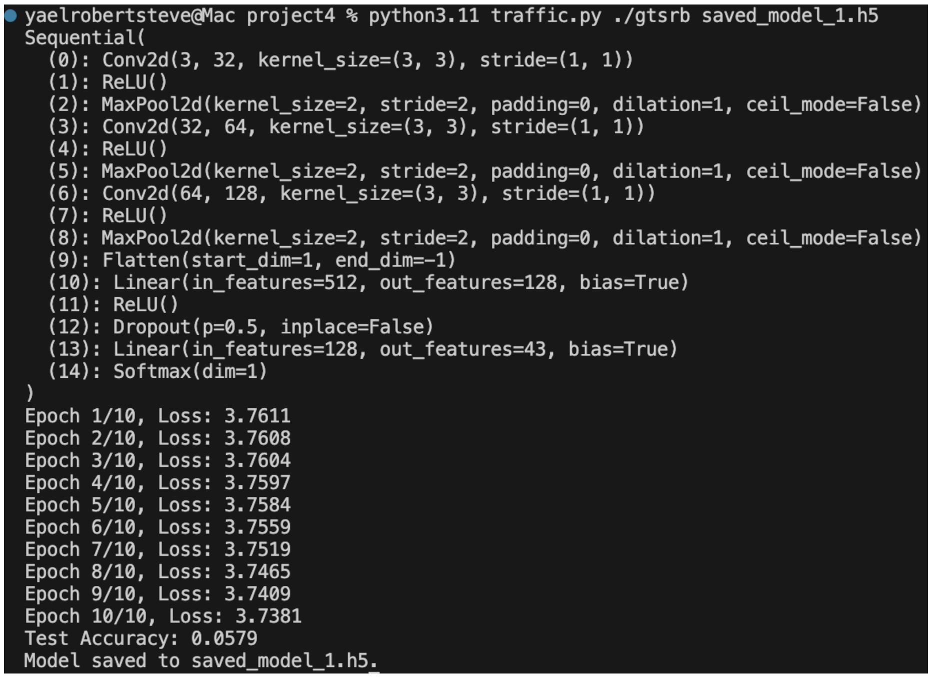
Trial 1

Keeping original values and a VGG style CNN, I started my experimentation!

```
EPOCHS = 10
IMG_WIDTH = 30
IMG_HEIGHT = 30
NUM_CATEGORIES = 43
TEST_SIZE = 0.4
```

```
# Set up optimizer and loss function for PyTorch
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = torch.nn.CrossEntropyLoss()
```

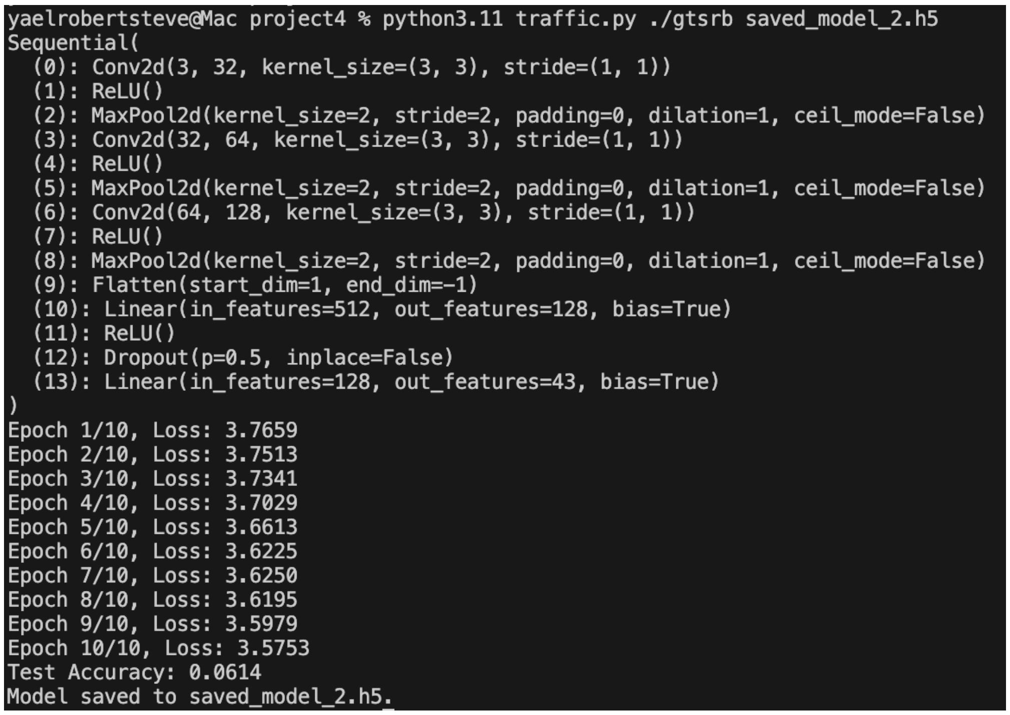
```
model = torch.nn.Sequential(
    torch.nn.Conv2d(3, 32, kernel_size=3),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2),
    torch.nn.Conv2d(32, 64, kernel_size=3),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2),
    torch.nn.Conv2d(64, 128, kernel_size=3),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2),
    torch.nn.Flatten(),
    torch.nn.Linear(128 * 2 * 2, 128),
    torch.nn.ReLU(),
    torch.nn.Dropout(0.5),
    torch.nn.Linear(128, NUM_CATEGORIES),
    torch.nn.Softmax(dim=1)
)
```



```
yaelrobertsteve@Mac project4 % python3.11 traffic.py ./gtsrb saved_model_1.h5
Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=512, out_features=128, bias=True)
  (11): ReLU()
  (12): Dropout(p=0.5, inplace=False)
  (13): Linear(in_features=128, out_features=43, bias=True)
  (14): Softmax(dim=1)
)
Epoch 1/10, Loss: 3.7611
Epoch 2/10, Loss: 3.7608
Epoch 3/10, Loss: 3.7604
Epoch 4/10, Loss: 3.7597
Epoch 5/10, Loss: 3.7584
Epoch 6/10, Loss: 3.7559
Epoch 7/10, Loss: 3.7519
Epoch 8/10, Loss: 3.7465
Epoch 9/10, Loss: 3.7409
Epoch 10/10, Loss: 3.7381
Test Accuracy: 0.0579
Model saved to saved_model_1.h5.
```

Trial 2

Since we are applying torch.nn.CrossEntropyLoss() Softmax is called internally. So, experimenting by removing additional Softmax layer from the neural network.

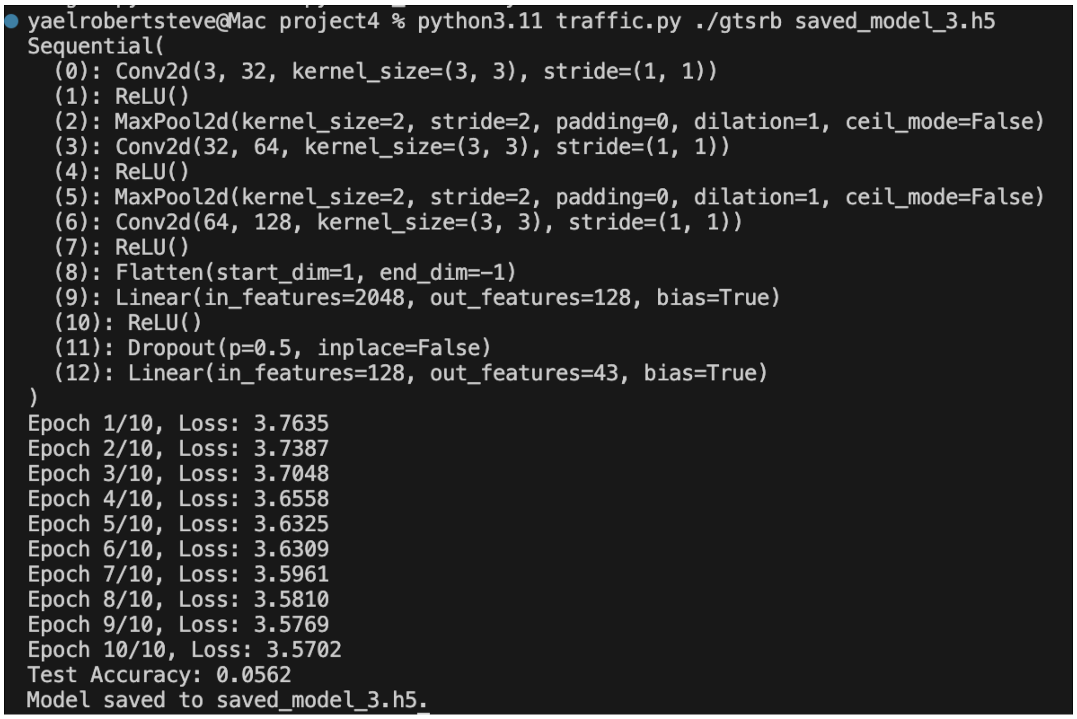


```
yaelrobertsteve@Mac project4 % python3.11 traffic.py ./gtsrb saved_model_2.h5
Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=512, out_features=128, bias=True)
  (11): ReLU()
  (12): Dropout(p=0.5, inplace=False)
  (13): Linear(in_features=128, out_features=43, bias=True)
)
Epoch 1/10, Loss: 3.7659
Epoch 2/10, Loss: 3.7513
Epoch 3/10, Loss: 3.7341
Epoch 4/10, Loss: 3.7029
Epoch 5/10, Loss: 3.6613
Epoch 6/10, Loss: 3.6225
Epoch 7/10, Loss: 3.6250
Epoch 8/10, Loss: 3.6195
Epoch 9/10, Loss: 3.5979
Epoch 10/10, Loss: 3.5753
Test Accuracy: 0.0614
Model saved to saved_model_2.h5.
```

Test accuracy slightly improved by 0.0035 units

Trail 3

3 MaxPool layers are crushing the image down so aggressively that spatial information is lost before the linear layers can see it. Removing the third MaxPool2d layer to let the Linear layer see a slightly larger feature map.

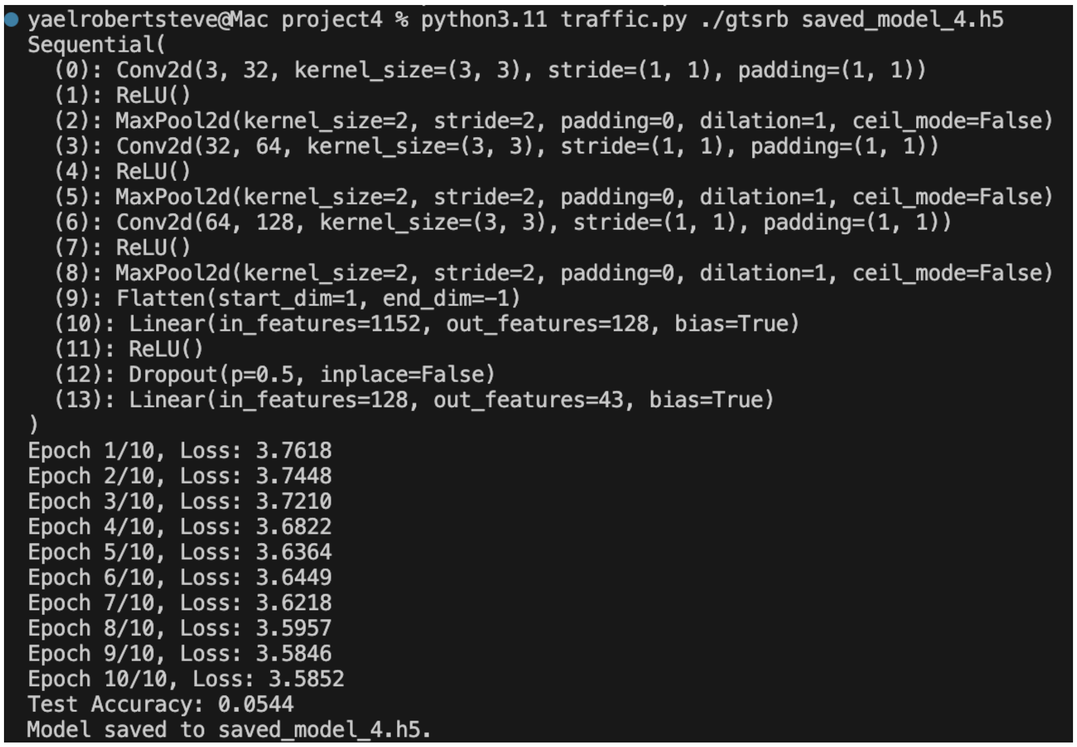


```
● yaelrobertsteve@Mac project4 % python3.11 traffic.py ./gtsrb saved_model_3.h5
  Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (7): ReLU()
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=2048, out_features=128, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.5, inplace=False)
    (12): Linear(in_features=128, out_features=43, bias=True)
  )
  Epoch 1/10, Loss: 3.7635
  Epoch 2/10, Loss: 3.7387
  Epoch 3/10, Loss: 3.7048
  Epoch 4/10, Loss: 3.6558
  Epoch 5/10, Loss: 3.6325
  Epoch 6/10, Loss: 3.6309
  Epoch 7/10, Loss: 3.5961
  Epoch 8/10, Loss: 3.5810
  Epoch 9/10, Loss: 3.5769
  Epoch 10/10, Loss: 3.5702
  Test Accuracy: 0.0562
  Model saved to saved_model_3.h5.
```

Test accuracy slightly lower than the trial #1

Trial 4

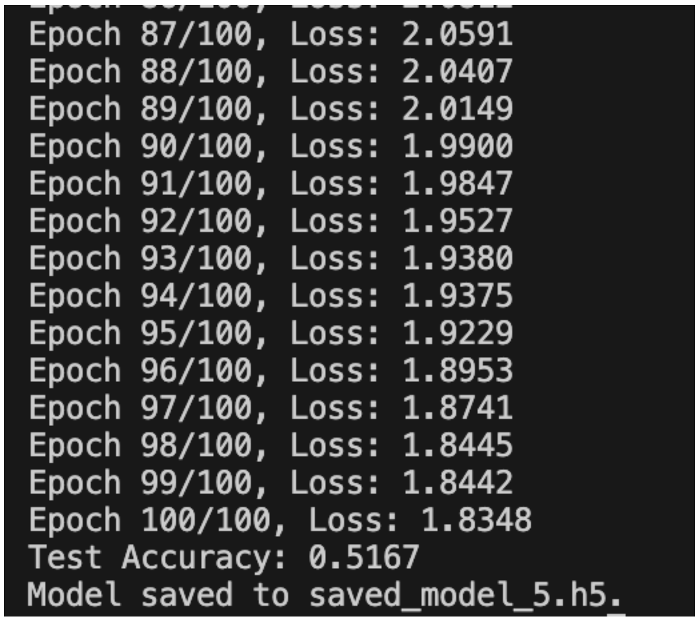
Adding padding to all 3 layers and 3rd layer has MaxPool layer. Because padding=1 prevented the image from shrinking during the convolution steps, we have enough pixels left to do the 3rd Pool and still end up with a valid 3x3 output.


```
yaelrobertsteve@Mac project4 % python3.11 traffic.py ./gtsrb saved_model_4.h5
Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=1152, out_features=128, bias=True)
  (11): ReLU()
  (12): Dropout(p=0.5, inplace=False)
  (13): Linear(in_features=128, out_features=43, bias=True)
)
Epoch 1/10, Loss: 3.7618
Epoch 2/10, Loss: 3.7448
Epoch 3/10, Loss: 3.7210
Epoch 4/10, Loss: 3.6822
Epoch 5/10, Loss: 3.6364
Epoch 6/10, Loss: 3.6449
Epoch 7/10, Loss: 3.6218
Epoch 8/10, Loss: 3.5957
Epoch 9/10, Loss: 3.5846
Epoch 10/10, Loss: 3.5852
Test Accuracy: 0.0544
Model saved to saved_model_4.h5.
```

Test accuracy even lower than the trial #3

Trial 5

Increasing epochs to 100 from 10 to move from underfitting to learning mode and keeping same model as Trial 4



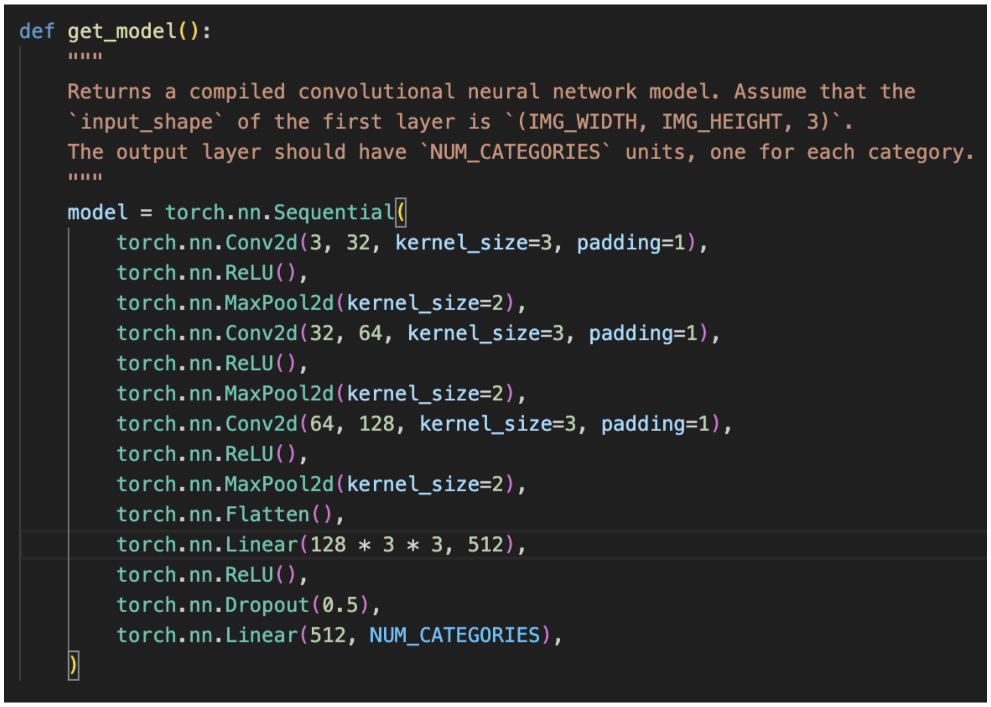
```
Epoch 87/100, Loss: 2.0591
Epoch 88/100, Loss: 2.0407
Epoch 89/100, Loss: 2.0149
Epoch 90/100, Loss: 1.9900
Epoch 91/100, Loss: 1.9847
Epoch 92/100, Loss: 1.9527
Epoch 93/100, Loss: 1.9380
Epoch 94/100, Loss: 1.9375
Epoch 95/100, Loss: 1.9229
Epoch 96/100, Loss: 1.8953
Epoch 97/100, Loss: 1.8741
Epoch 98/100, Loss: 1.8445
Epoch 99/100, Loss: 1.8442
Epoch 100/100, Loss: 1.8348
Test Accuracy: 0.5167
Model saved to saved_model_5.h5.
```

Improves accuracy from the best till now [trial 2: 0.0614] to 0.5167. That is 0.4553 increase!

Trial 6

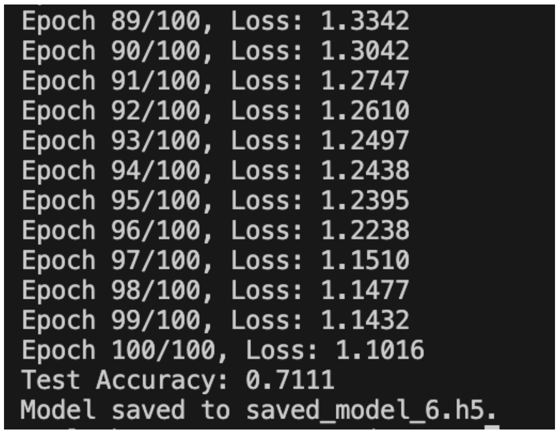
Increasing number of hidden layers from 128 to 512. 43 categories is a lot. 32 or 64 filters might not be enough "brain capacity" to memorize the difference between a "Pedestrian

Crossing" and "School Crossing."



```python
def get_model():
    """
    Returns a compiled convolutional neural network model. Assume that the
    `input_shape` of the first layer is `(IMG_WIDTH, IMG_HEIGHT, 3)`.
    The output layer should have `NUM_CATEGORIES` units, one for each category.
    """
    model = torch.nn.Sequential(
        torch.nn.Conv2d(3, 32, kernel_size=3, padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=2),
        torch.nn.Conv2d(32, 64, kernel_size=3, padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=2),
        torch.nn.Conv2d(64, 128, kernel_size=3, padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=2),
        torch.nn.Flatten(),
        torch.nn.Linear(128 * 3 * 3, 512),
        torch.nn.ReLU(),
        torch.nn.Dropout(0.5),
        torch.nn.Linear(512, NUM_CATEGORIES),
    )
```

The epochs reduced the loss faster than the previous trial.

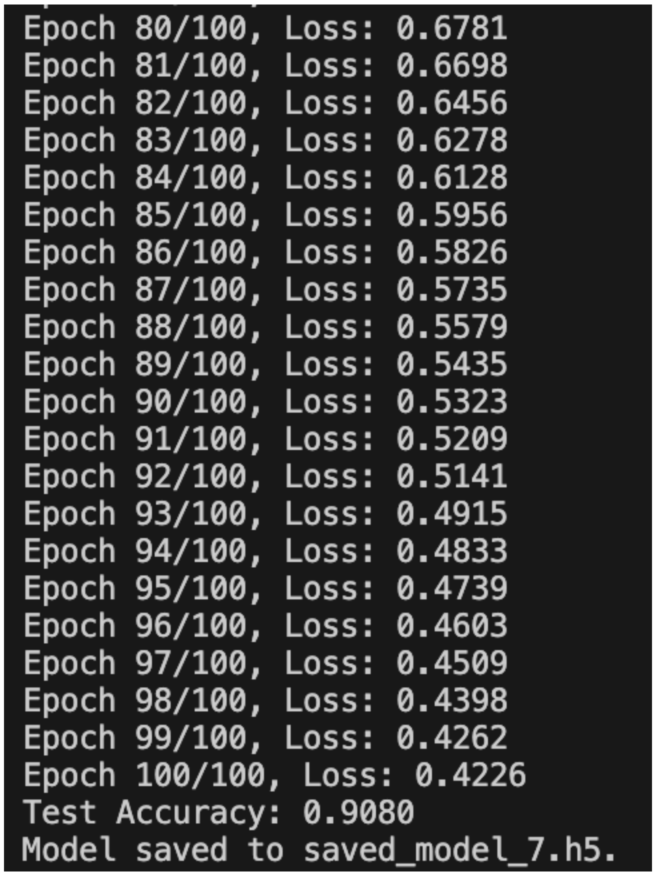


```
Epoch 89/100, Loss: 1.3342
Epoch 90/100, Loss: 1.3042
Epoch 91/100, Loss: 1.2747
Epoch 92/100, Loss: 1.2610
Epoch 93/100, Loss: 1.2497
Epoch 94/100, Loss: 1.2438
Epoch 95/100, Loss: 1.2395
Epoch 96/100, Loss: 1.2238
Epoch 97/100, Loss: 1.1510
Epoch 98/100, Loss: 1.1477
Epoch 99/100, Loss: 1.1432
Epoch 100/100, Loss: 1.1016
Test Accuracy: 0.7111
Model saved to saved_model_6.h5.
```

The accuracy improved to 0.7111, which is an increase of 0.1944 from the previous trial.

Trial 7

Removing 3rd MaxPool to give the Linear Layer more information to work with

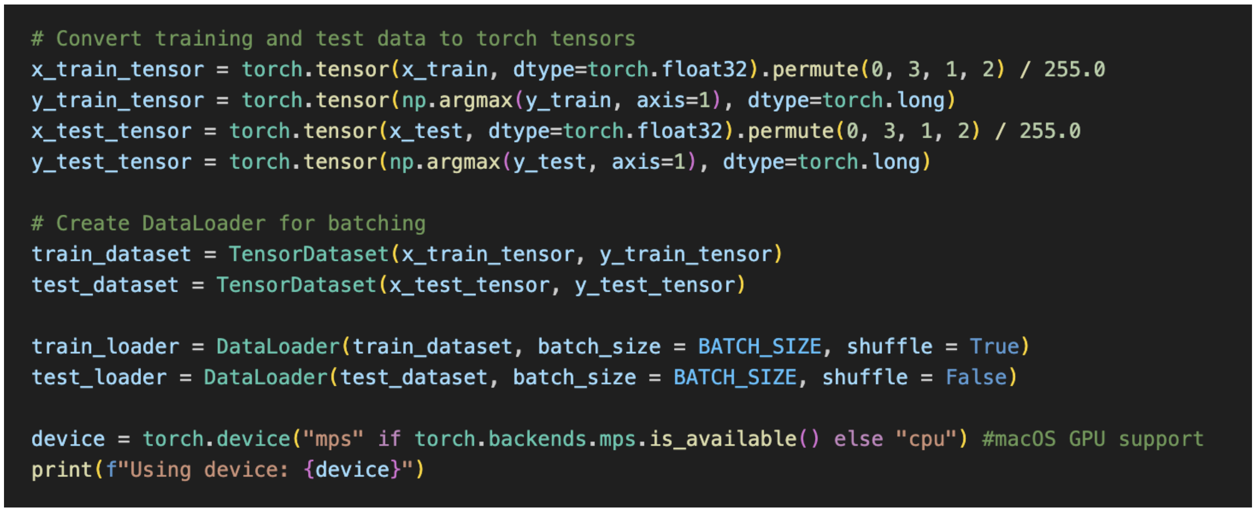


```
Epoch 80/100, Loss: 0.6781
Epoch 81/100, Loss: 0.6698
Epoch 82/100, Loss: 0.6456
Epoch 83/100, Loss: 0.6278
Epoch 84/100, Loss: 0.6128
Epoch 85/100, Loss: 0.5956
Epoch 86/100, Loss: 0.5826
Epoch 87/100, Loss: 0.5735
Epoch 88/100, Loss: 0.5579
Epoch 89/100, Loss: 0.5435
Epoch 90/100, Loss: 0.5323
Epoch 91/100, Loss: 0.5209
Epoch 92/100, Loss: 0.5141
Epoch 93/100, Loss: 0.4915
Epoch 94/100, Loss: 0.4833
Epoch 95/100, Loss: 0.4739
Epoch 96/100, Loss: 0.4603
Epoch 97/100, Loss: 0.4509
Epoch 98/100, Loss: 0.4398
Epoch 99/100, Loss: 0.4262
Epoch 100/100, Loss: 0.4226
Test Accuracy: 0.9080
Model saved to saved_model_7.h5.
```

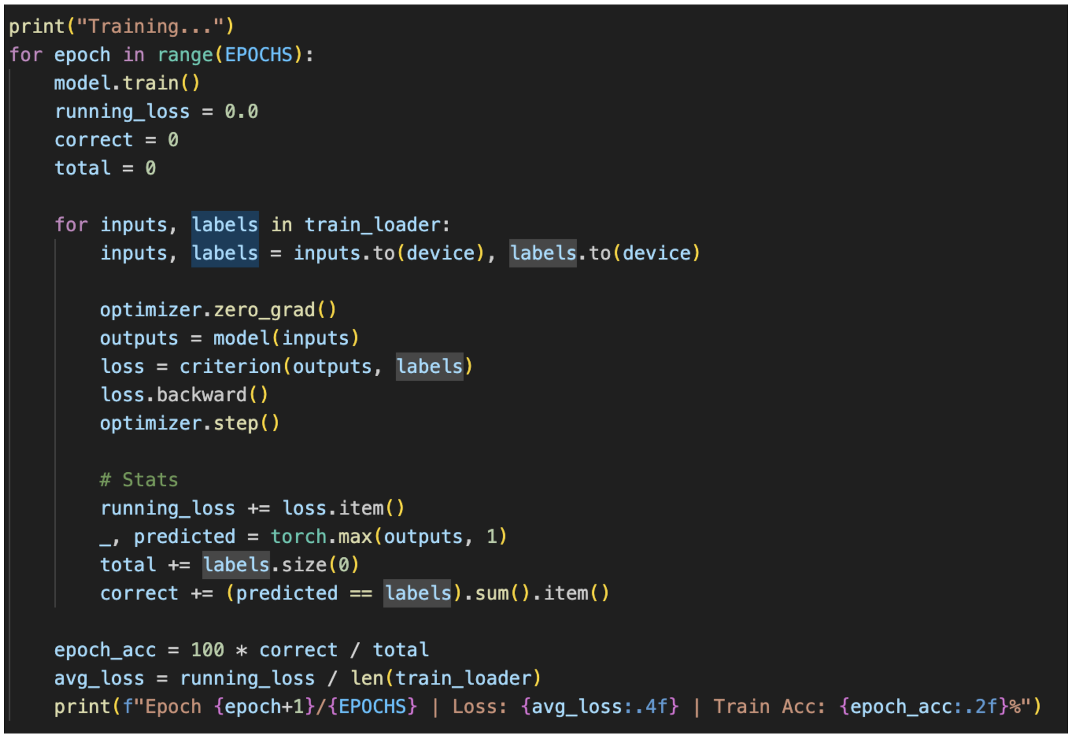
Accuracy improved to 0.9080 ie 90.8% on test dataset of 0.4 portion of GTSRB dataset

Trial 8

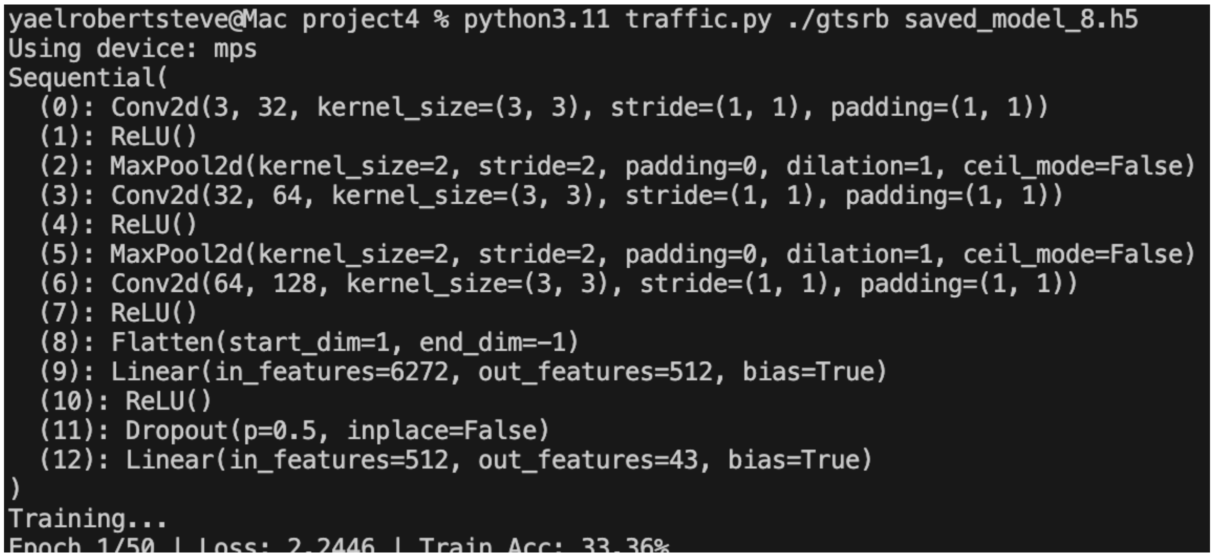
Updated code for memory safe utilization, and it now pushes 32 images at a time [PyTorch's DataLoaders]. This makes the model learn faster (more updates per epoch) and prevents crashing. Reduced the epochs to 50, used MPS instead of CPU and showed more statistics [training accuracy and testing accuracy in percentages]



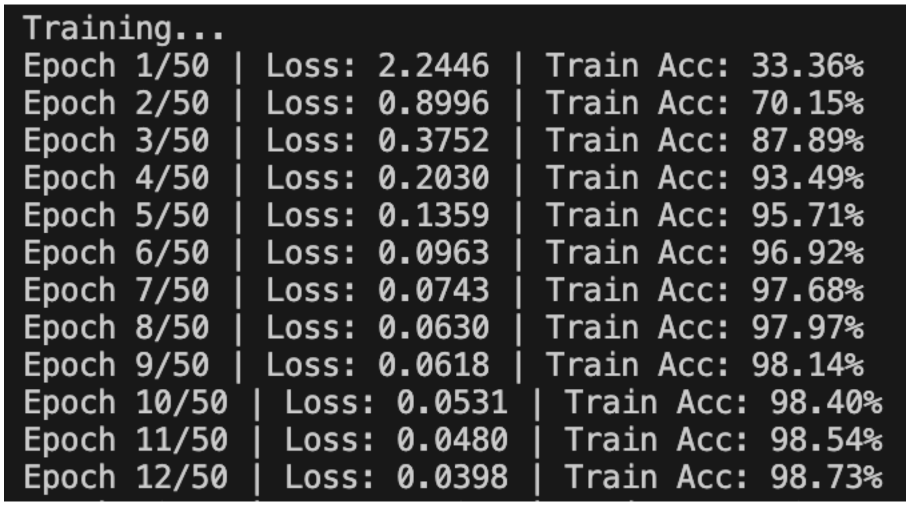
```python
# Convert training and test data to torch tensors
x_train_tensor = torch.tensor(x_train, dtype=torch.float32).permute(0, 3, 1, 2) / 255.0
y_train_tensor = torch.tensor(np.argmax(y_train, axis=1), dtype=torch.long)
x_test_tensor = torch.tensor(x_test, dtype=torch.float32).permute(0, 3, 1, 2) / 255.0
y_test_tensor = torch.tensor(np.argmax(y_test, axis=1), dtype=torch.long)

# Create DataLoader for batching
train_dataset = TensorDataset(x_train_tensor, y_train_tensor)
test_dataset = TensorDataset(x_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size = BATCH_SIZE, shuffle = True)
test_loader = DataLoader(test_dataset, batch_size = BATCH_SIZE, shuffle = False)

device = torch.device("mps" if torch.backends.mps.is_available() else "cpu") #macOS GPU support
print(f"Using device: {device}")
```



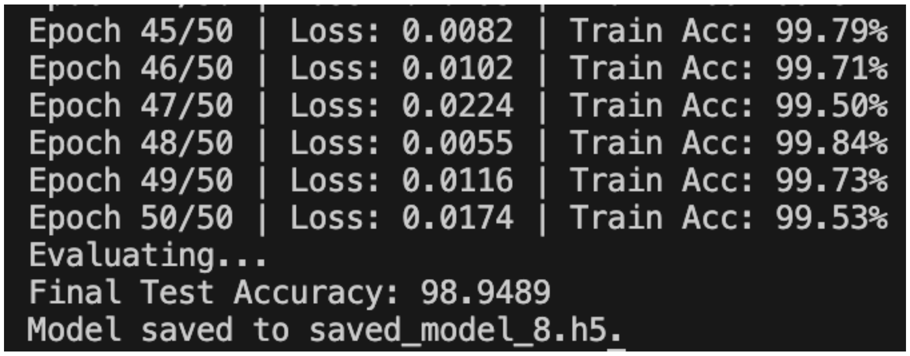
```python
print("Training...")
for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Stats
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_acc = 100 * correct / total
    avg_loss = running_loss / len(train_loader)
    print(f"Epoch {epoch+1}/{EPOCHS} | Loss: {avg_loss:.4f} | Train Acc: {epoch_acc:.2f}%")
```



```
yaelrobertsteve@Mac project4 % python3.11 traffic.py ./gtsrb saved_model_8.h5
Using device: mps
Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU()
  (8): Flatten(start_dim=1, end_dim=-1)
  (9): Linear(in_features=6272, out_features=512, bias=True)
  (10): ReLU()
  (11): Dropout(p=0.5, inplace=False)
  (12): Linear(in_features=512, out_features=43, bias=True)
)
Training...
Epoch 1/50 | Loss: 2.2446 | Train Acc: 33.36%
```



```
Training...
Epoch 1/50 | Loss: 2.2446 | Train Acc: 33.36%
Epoch 2/50 | Loss: 0.8996 | Train Acc: 70.15%
Epoch 3/50 | Loss: 0.3752 | Train Acc: 87.89%
Epoch 4/50 | Loss: 0.2030 | Train Acc: 93.49%
Epoch 5/50 | Loss: 0.1359 | Train Acc: 95.71%
Epoch 6/50 | Loss: 0.0963 | Train Acc: 96.92%
Epoch 7/50 | Loss: 0.0743 | Train Acc: 97.68%
Epoch 8/50 | Loss: 0.0630 | Train Acc: 97.97%
Epoch 9/50 | Loss: 0.0618 | Train Acc: 98.14%
Epoch 10/50 | Loss: 0.0531 | Train Acc: 98.40%
Epoch 11/50 | Loss: 0.0480 | Train Acc: 98.54%
Epoch 12/50 | Loss: 0.0398 | Train Acc: 98.73%
```

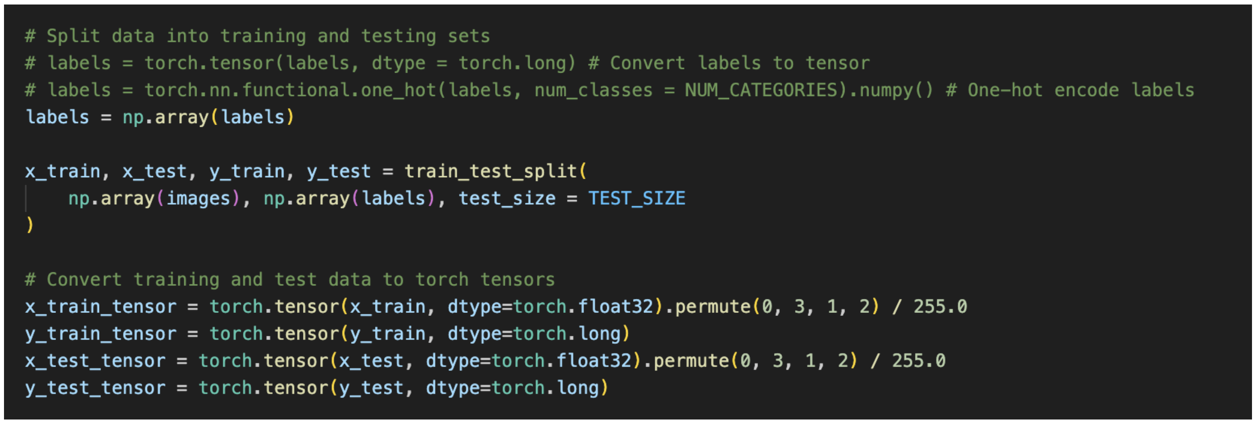


```
Epoch 45/50 | Loss: 0.0082 | Train Acc: 99.79%
Epoch 46/50 | Loss: 0.0102 | Train Acc: 99.71%
Epoch 47/50 | Loss: 0.0224 | Train Acc: 99.50%
Epoch 48/50 | Loss: 0.0055 | Train Acc: 99.84%
Epoch 49/50 | Loss: 0.0116 | Train Acc: 99.73%
Epoch 50/50 | Loss: 0.0174 | Train Acc: 99.53%
Evaluating...
Final Test Accuracy: 98.9489
Model saved to saved_model_8.h5.
```

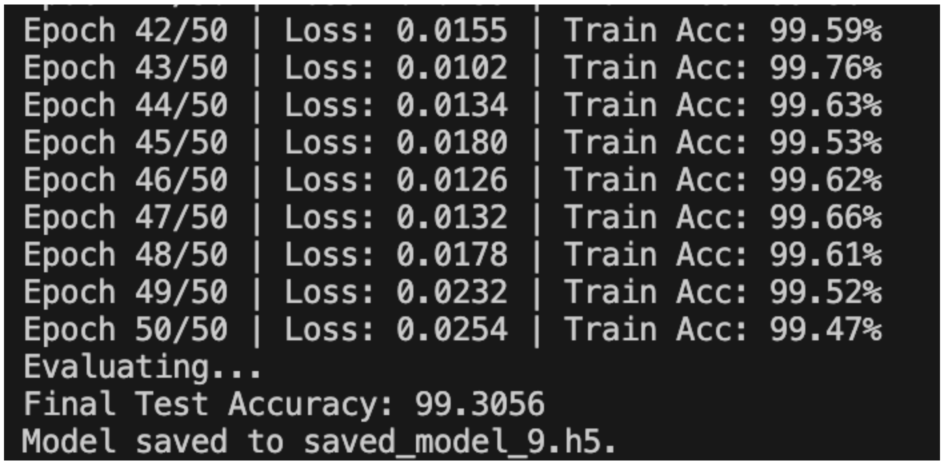
This accuracy is the highest till now. 98.9489%

Trial 9

PyTorch's CrossEntropyLoss prefers simple class integers. Code updated to avoid one-hot encoding and argmax. Thus, we avoid wasting CPU cycles which happen when converting numbers to vectors and back again.



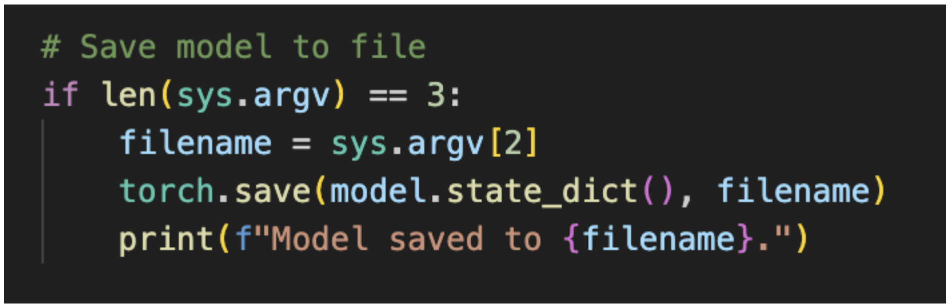
```python
# Split data into training and testing sets
# labels = torch.tensor(labels, dtype = torch.long) # Convert labels to tensor
# labels = torch.nn.functional.one_hot(labels, num_classes = NUM_CATEGORIES).numpy() # One-hot encode labels
labels = np.array(labels)

x_train, x_test, y_train, y_test = train_test_split(
    np.array(images), np.array(labels), test_size = TEST_SIZE
)

# Convert training and test data to torch tensors
x_train_tensor = torch.tensor(x_train, dtype=torch.float32).permute(0, 3, 1, 2) / 255.0
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
x_test_tensor = torch.tensor(x_test, dtype=torch.float32).permute(0, 3, 1, 2) / 255.0
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```



```
Epoch 42/50 | Loss: 0.0155 | Train Acc: 99.59%
Epoch 43/50 | Loss: 0.0102 | Train Acc: 99.76%
Epoch 44/50 | Loss: 0.0134 | Train Acc: 99.63%
Epoch 45/50 | Loss: 0.0180 | Train Acc: 99.53%
Epoch 46/50 | Loss: 0.0126 | Train Acc: 99.62%
Epoch 47/50 | Loss: 0.0132 | Train Acc: 99.66%
Epoch 48/50 | Loss: 0.0178 | Train Acc: 99.61%
Epoch 49/50 | Loss: 0.0232 | Train Acc: 99.52%
Epoch 50/50 | Loss: 0.0254 | Train Acc: 99.47%
Evaluating...
Final Test Accuracy: 99.3056
Model saved to saved_model_9.h5.
```

Accuracy all time high 99.3056%

I later updated all my saved models with .pt extension because pytorch saves as .pt and at that point I had saved it with the wrong extension although the file was a .pt file because of pytorch's code



```python
# Save model to file
if len(sys.argv) == 3:
    filename = sys.argv[2]
    torch.save(model.state_dict(), filename)
    print(f"Model saved to {filename}.")
```

And updated the requirements file as needed.

GTSRB Traffic Sign Classification: Algorithm and Model Build Report

1) Problem & Dataset

- **Task:** Multiclass image classification of German traffic signs (GTSRB), with **43 categories**.
- **Input format expected by the code:** RGB images resized to **30×30**.
- **Directory layout:** a root data directory (e.g., gtsrb/) containing subfolders named 0, 1, ..., 42, each with images for that class.

2) Algorithm Introduction

This project implements a **Convolutional Neural Network (CNN)** classifier in **PyTorch**. The CNN learns hierarchical visual features (edges → textures → shapes) from traffic sign images:

- Convolutions extract spatial patterns using small kernels.
- ReLU provides non-linearity.

- Max pooling down-samples to build translational invariance and reduce compute.
- Fully-connected layers map the learned features to class logits.
- Training minimizes **cross-entropy loss** with the **Adam** optimizer.

CNNs are the de facto standard for 2D image recognition tasks like traffic sign classification due to their parameter sharing and locality properties.

3) Data Loading & Preprocessing (from traffic.py)

- **Loading:** Images are read with OpenCV from each class directory.
- **Resize:** Each image is resized to **30×30**.
- **Split:** Uses train_test_split with TEST_SIZE = 0.4 (i.e., 60% train, 40% test).
- **Tensor conversion:** Arrays are converted to PyTorch tensors and permuted to **NCHW** ((N, 3, 30, 30)).
- **Normalization:** Pixel values are scaled to [0, 1] by dividing by 255.0.

4) Model Architecture

Defined in get_model() as a torch.nn.Sequential network:

```
Input: (3, 30, 30)
[Conv2d 3→32, k=3, pad=1] → ReLU → MaxPool2d(2)
[Conv2d 32→64, k=3, pad=1] → ReLU → MaxPool2d(2)
[Conv2d 64→128, k=3, pad=1] → ReLU
Flatten
Linear 128×7×7 (=6272) → 512 → ReLU → Dropout(p=0.5)
Linear 512 → 43 logits
```

- Output logits are passed to CrossEntropyLoss, which applies log_softmax internally.
- Suitable for small inputs (30×30); two pooling layers reduce spatial size to 7×7.

Approximate parameter counts:

- Conv(3→32, k=3): ~896
- Conv(32→64, k=3): ~18,496
- Conv(64→128, k=3): ~73,856
- Dense(6272→512): ~3,211,776

- Dense(512→43): ~22,059
- **Total:** ~3.33M parameters

5) Training Configuration

- **Epochs:** EPOCHS = 50
- **Batch size:** BATCH_SIZE = 32
- **Optimizer:** Adam(lr=0.001)
- **Loss:** CrossEntropyLoss
- **Device:** Uses Apple Silicon **MPS** if available, otherwise CPU (torch.device("mps" if available else "cpu")).
- **Metrics:** Prints per-epoch training loss and accuracy; after training, reports final test accuracy.

6) Evaluation

- Evaluation is performed on the held-out test split (40%).

7) Saving & Loading the Model

- The script saves the **state dict** when a filename is provided as the second CLI argument:
  - torch.save(model.state_dict(), filename)
  - Recommended extension: **.pt** or **.pth** (e.g., model.pt).

8) How the Model is Built

1. Read and resize images from gtsrb/<class_id>/*.ppm into 30×30 arrays.
2. Split into train/test (0.6/0.4).
3. Convert to tensors, permute to NCHW, and scale to [0, 1].
4. Construct the CNN as defined above.
5. Train for 50 epochs with Adam and cross-entropy, using batches of 32.
6. Evaluate on the test split and print accuracy.
7. Optionally save model weights with torch.save.

9) Reproducibility & Running

Dependencies

The script uses **PyTorch**, **OpenCV**, and **scikit-learn**. Install them (MPS works on macOS 12+ and recent PyTorch - I used Python 3.11.14):

```
python -m pip install --upgrade pip
```

```
pip install -r requirements.txt
```